EXHIBIT J

A United States Provisional Patent Application

for

SafeFox: A Safe Lightweight Virtual Browsing Environment

by

Anup K. Ghosh
Sushil Jajodia
Jiang Wang
Yih Huang

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

This invention was made with government support under Army Contract #W31P4Q-07-C-0244 awarded by Defense Advanced Research Projects Agency. The government has certain rights in the invention.

SafeFox: a Safe Lightweight Virtual Browsing Environment

Abstract

The browser has become a popular attack vector for implanting code on computer operating systems. Equally critical, important sessions, such as online banking, must be protected from cross-site attacks from other concurrent sessions. In this work we describe an approach using lightweight virtualization to create a safe browsing environment, called SafeFox, to protect both the host and important browsing sessions from malicious Web content. With SafeFox, the browser runs in its own virtual environment (VE) in its own process namespace, file system, and IP address; furthermore, when browsing to a secure bookmarked site SafeFox automatically creates a new isolated lightweight virtual environment (VE) for the secure bookmarked site. In this paper, we present the architecture for SafeFox and demonstrate its low-overhead approach while analyzing its security properties. While the native platform of SafeFox is Linux, we have created a SafeFox virtual appliance to run on multiple platforms, including Windows™.

1. Introduction

Currently, the Web browser has become the most popular and convenient attack vector, posing significant risk to end-user's computing environment [17]. With the advent of Web 2.0 sites where content is populated by untrusted users, the browser increasingly executes active content derived from untrusted sources even on trusted web sites, such as Facebook. Furthermore, in current web browser design, a single browser instance is used for multiple, concurrent tasks: reading news, on-line banking, on-line shopping, or playing on-line games. This design is convenient but leads to potential security risks between different tasks.

A modern browser typically launches a group of processes and threads for use by the user. When the user opens multiple browser windows and/or tabs, concurrent sessions are collectively managed by these browser processes/threads. This management task inevitably involves communication and information sharing among the processes and threads. These collaborating processes and threads form a *browser instance*. Furthermore, these browser processes/threads run in the same process namespace and file system as other programs on the host. A single program flaw in the browser code can allow untrusted Web content to exploit other browser sessions as well as other programs and data on the host.

Attacks through the browser can take one of two forms. The first type of attack compromises the user's system. Drive-by downloads of malware and its execution is an example of such attacks. Here, code on a Web site exploits a vulnerability in the browser to write a file to the file system. The same or another vulnerability is then used to run the dropped file. In a safe browsing environment, the user's computer system must be protected from malicious Web contents, hereinafter referred to as *host* protection. The second type of attack, such as cross-site scripting attacks [1,2], remains resident in the process memory space attempting to interfere with other concurrent browsing sessions. For such attacks, important online sessions, such as online banking, must be shielded from interference from other concurrent sessions. This will be referred to as *session* protection.

Previously, many forms of browser protection techniques have been extensively investigated, ranging from process-level isolation, browser privilege restrictions to the deployment of virtualization technologies. Many research efforts in browsing security, including this one, argue that the highest level of protection can only be achieved by running browsers in isolated, virtualized environments, rather than directly on the host.

Several virtual appliances built for safe browsing essentially run commodity browsers within virtual machines (VM) [36]. In this way, the browser (instance) runs within the VM and the host is protected by the VM. However *session* protection still relies on security mechanisms of commodity browsers, typically based on privilege restrictions and/or processes/threads sandboxing. There exists prior research work on using para-virtualization for *both* host and session protections [4]. Such an approach amounts to dedicating one VM to each Web session. While its advantages in session isolation are many, the overhead of running a full virtual machine for each browser session is significant and potentially unacceptable.

To address the requirements of host protection, session protection, and low overhead, we employ lightweight virtualization (also known as OS-level virtualization) to create a safe browsing environment, called SafeFox. In SafeFox, when the user clicks on

the Firefox™ icon or menu item, a new lightweight virtual environment (VE) is created to execute a browser instance in the VE. The VE is assigned its own process namespace, IP address, and file system. In addition, users of SafeFox can select important Web sites as "*secure bookmarks*." When a user visits a secure bookmarked site, SafeFox creates another VE on demand to execute a new browser instance. If the user visits more than one secure bookmarked site simultaneously, multiple browser VEs are created, one for each site. Each VE runs independently, and in fact has no knowledge of the others' existence.

A high-level view of the SafeFox environment is shown in Figure 1. The environment is comprised of a customized Linux kernel and a couple of background control daemons that issue commands to create VEs, launch Firefox from within VEs, destroy VEs when the user terminates corresponding Firefox instances, and direct traffic with secure-bookmarked URLs traffic to respective VEs. In spite of its apparent complexity, the user interface is identical to running Firefox natively – a double click on the desktop icon starts the environment and a browser window shows up. We show in Figure 1 where a SafeFox user has created one general-purpose browser instance and concurrently visits two secure-bookmark sites. As seen in the figure, secure bookmarked sites have their dedicated browser instances and all three browser instances are running in three separate VEs.

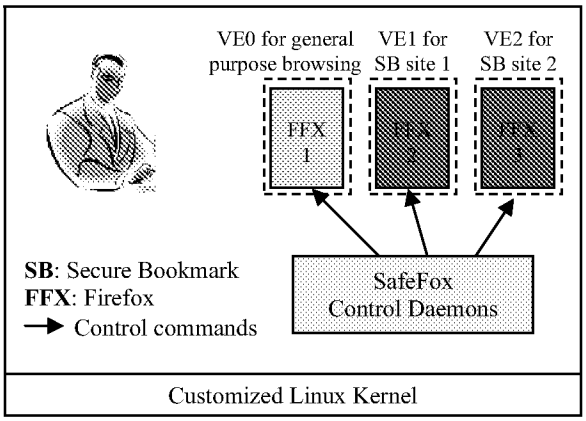


Figure 1. The SafeFox browsing environment

The native platform of SafeFox is Linux. This limits its users to the Linux community, which constitutes less than 1% of client-end users. To bring the benefits of SafeFox to the users of other platforms, we have also created a SafeFox VMware™ virtual appliance.

The primary contributions of SafeFox include the following.

- The use of lightweight virtualization for safe browsing. All browsing activities take place inside lightweight VEs, rather than the host.
- Isolation of important Web sessions pertaining to secure bookmarked sites using separate browser instances running in their respective VEs.

We will show in our performance evaluation results that SafeFox performs with minimal overhead so that multiple browser instances can be launched without burdening the host significantly. Since many Windows™ users stand to benefit from SafeFox, we will describe SafeFox in the context where it is executed in a virtual machine such as VMWare's VMPlayer™.

In the remainder of this paper, Section 2 reviews the building-block technologies used to create SafeFox. In Section 3, we introduce the features of SafeFox from a user's perspective. In Section 4, we describe the design and implementation of SafeFox. We present in Section 5 performance results and security analysis of SafeFox. Related work and conclusion are given in Sections 6 and 7, respectively.

2. Building-block Technologies

2.1. Full/para virtualization

As seen in Figure 2, full/para virtualization runs full operating systems, including their kernels, in virtual machines. Either a hypervisor-only layer on top of the hardware or a host operating system with hypervisor support manages running VMs. A non-exhaustive list of present virtualization technologies includes [7, 8, 9, 10].

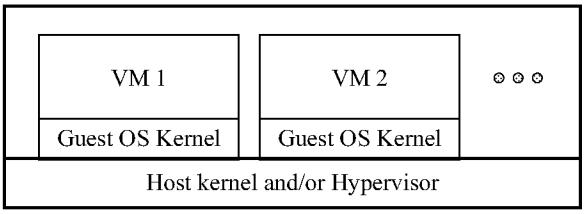


Figure 2. Full/para virtualizations

Full/para virtualization provides strong isolation among VMs and for the host. They also support heterogeneous guest operating systems. However, these technologies impose the overhead of two-level resource management: first by the host kernel and/or hypervisor, second by guest operating systems. This introduces significant overhead especially when used to run multiple browser instances, each in its own VM, for session protection.

2.2. Lightweight virtualization

In contrast with full virtualization, the lightweight virtualization solution uses a single kernel for the host and virtual environments (see Figure 3). All virtualized environments are directly managed by the host kernel. Despite the lack of its own kernel, a lightweight VE behaves externally just like full virtual machines or physical hosts. Without the overhead of two-level management, lightweight virtualization imposes much less performance penalty. Examples of lightweight virtualization technologies can be found in [11, 12, 13, 14, 15]. SafeFox adopts OpenVZ [15] as its lightweight virtualization solution.

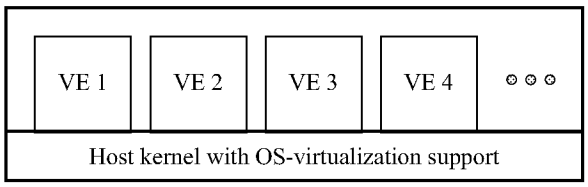


Figure 3. Lightweight/OS virtualization

One important restriction of lightweight virtualization is that the OS in a VE must be based on one similar to the host OS. For instance, an OpenVZ-based RedHat Linux can run any Linux varieties, such as Ubuntu, as its guest VEs. However, it does not support Windows VEs.

2.3. Creating VE on demand

We have previously developed and published a novel method to create lightweight VEs on demand [19]. Here we briefly review the method, shown in Figure 4.

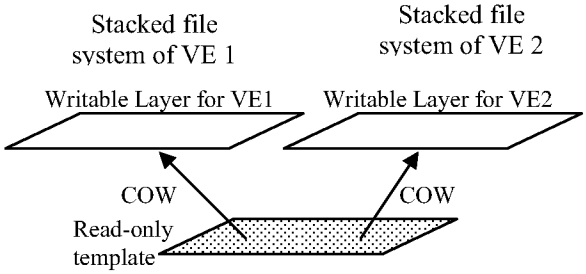


Figure 4. Template sharing among VEs

First we create a read-only template VE. As shown in Figure 4, a new VE is created by adding a writeable layer on top of read-only template, using a stackable file system [16]. We show two such stacks in the figure. The stack pertaining to a VE is then mounted as the root of the file system of that VE. When the VE performs the first write on a file in the template, a copy-on-write (COW) operation creates a new version of the file in the writable layer, which is used by the VE for later reads and writes on the file.

After the creation of its file system, the VE is configured with an IP address and subsequently booted up. In this way, all VEs start with the same state as the template, with exception of IP addresses. They will drift apart over time as applications in respective VEs modify their respective file systems.

The consequences of the above VE construction technique can be summarized as follows. In SafeFox, there is only one copy of the Firefox binary and associated libraries in the read-only template, regardless the number of active browser VEs; hence its economy in disk space. Moreover there is only one copy of Firefox binary and its libraries in the *physical* memory that are mapped to the virtual memory spaces of concurrent browser instances in VEs; hence its memory economy. The combination of the above two factors in SafeFox translates to its lightweight in terms of hard disk and memory consumption.

3. The SafeFox Browsing Environment

For the highest level of browsing protection, every concurrent Web session should be executed in its own isolated browser instances. This extreme approach, however, creates practical difficulties. For instance, creating a separate browser window for each different Web site will consume significant desktop real estate – one reason tabbed browsing emerged as a feature. Also, a user will want one central browser instance to manage bookmarks, add-ons, passwords, and so on, rather than managing them on all instances. To address such practical issues, SafeFox provides three browsing modes: the *master* mode, *secure-bookmark* mode and *private* mode. These modes are used for normal Web sites, for secure transaction Web sites and for private browsing.

The design principles of SafeFox are compatible with any browser implementations supported on the Linux platform. In our prototype implementation, we choose the popular and relatively-safe Mozilla Firefox browser. We will use the terms browser and Firefox interchangeably in the following discussions.

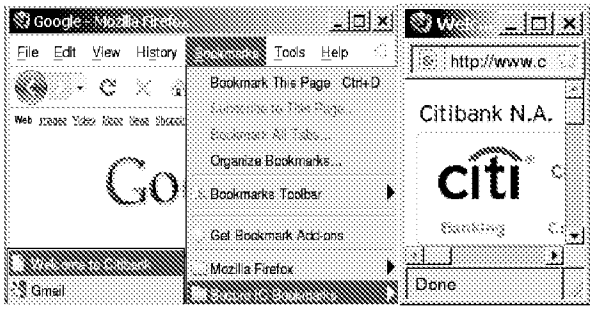
3.1. Master mode

When the user double clicks the SafeFox icon on the desktop, an instance of Firefox in the master mode is automatically launched if it does not already exist. The master browser supports all functions of a "normal" Firefox browser running directly on the host. It keeps browsing history, bookmarks, cookies, passwords, add-ons and so on, collectively referred to

as personalized browser *metadata* hereafter. There can be only one master browser at a time. If the user clicks on the desktop SafeFox icon again, a new Firefox tab will be created in the master Firefox. The master Firefox is used for typical Web browsing activities, such as news reading, forum discussion, blogging, social networking, playing YouTube videos, and so on.

3.2. Secure-bookmark mode

In the master Firefox, there is a pre-configured bookmark directory called "Secure IC Bookmarks" as shown in Figure 5(a). URLs stored in that directory are called *secure bookmarks*. Clicking on a secure bookmark in the master browser creates a new isolated browser instance in the secure-bookmark mode. Figure 5(b) shows a size-reduced version of the browser created from a secure bookmark. The new secure-bookmark Firefox instance is provided with the designated URL as an input parameter. It will then visit the URL (Citi™ in Figure 5(b)).



5(a). The master **5(b)**. A secure bookmark site

Figure 5. Launching a secure-bookmark mode browser

In the secure bookmark mode, the navigation bar is read-only and all top menus are disabled. Note in Figure 5(b) the absence of the normal top Firefox menu bar. In addition, the URL bar is set to read-only. The traffic with secure bookmarked site is redirected to the new browser instance. For each secure bookmarked site, there will be a corresponding directory on the host to preserve the metadata pertaining to that site. The secure bookmark mode is suitable for sites where the user requires secure transactions, such as online banking and shopping sites, and where other browser instances cannot spy or interfere with the transactions. A user adds or removes secure bookmarks through the standard Firefox bookmark management interface in the master Firefox.

We point out that, even though disabling the URL bar prevents the user from typing new URLs, the user may still follow links embedded in the secure sites. Some of those links may leave the designated secure site. SafeFox does not prevent this from happening. In general, enforcing a browser to stay in a (set of) Web site requires the service sites to provide per-service information and policy, a practice advocated in [4]. We will add the enforcement feature if or when the practice becomes common.

3.3. Private mode

In the SafeFox environment, there is a second icon named "Private Firefox" that is used to create a new browser instance executed in a newly created VE without any initial metadata. It consequently starts in an anonymous state where no personal data about the user is kept. A private browser instance has access to all browser functionalities. After the instance is terminated, however, its VE is destroyed and all metadata produced during that session are removed, thus maintaining privacy of the session from future observation or discovery. The user can create as many private and secure-bookmark browser instances as the memory of the system can support.

3.4. Miscellaneous discussions

File downloads and uploads to/from the host are supported in the master and secure-bookmark modes but not the private mode as the private mode has no persistent link to the desktop. Moreover, SafeFox restricts downloads and uploads to the Desktop directory only. While the restriction may cause minor inconvenience for the user, it significantly increases the security level of the browsing environment. This is a typical security-versus-convenience tradeoff.

Moreover, SafeFox browser instances in different modes are visually distinguished by different "SafeFox themes."

3.5. SafeFox as a virtual appliance

When SafeFox is running in a VMware virtual appliance, the user's experiences are identical. Browser metadata are preserved on the Windows hosts through the "shared folder" feature of VMware – the feature allows a Windows folder to be mapped to a Linux directory. Lastly, to avoid the long boot-up delays of guest operating systems, we use the snapshot feature of VMware to quickly restore the SafeFox VM to a point when the VM is ready to serve.

4. Architecture and Implementations

We describe in this section the architecture and a prototype implementation of SafeFox. The slight changes that we make to port SafeFox as virtual appliance will be discussed at the end of the section.

4.1. Prototype platform

As discussed in Section 2.3, SafeFox uses our previously developed method to quickly create VEs on demand. This method requires a customized kernel. Our prototype is Dell M4300 Laptop equipped a Core 2 Dual processor (2 cores) and 4 GB memory and configured with RedHat Enterprise Linux (RHEL) 5.3 as the operating system. We replaced standard RedHat kernel with a 2.6.24 kernel customized with OpenVZ [15] and Unionfs [16] patches. The VE template is based on CentOS Linux 5.3, a legally free version of RHEL 5.3. The reason of using the free version is legal concerns. Even though we have a legal license for RHEL 5 for the laptop, we are not certain whether we can create arbitrary numbers of RHEL VEs with only one license key. In addition, we configured the VEs to use OpenDNS [37] to mitigate the DNS attacks.

A Perl script `safefox.pl` is developed to construct a new VE file system as described in Section 2.3, allocate it an IP address, boot up the new VE, execute Firefox from within the VE, and project the new Firefox window to the desktop X server. When creating a master Firefox, a predetermined fixed IP address is assigned to the new VE. When creating VEs in other modes, IP addresses are dynamically allocated. For master mode or secure-bookmark mode VEs, the user's Desktop directory is "bind mounted" [18] into the VE's file system and hence accessible by these VEs. This is for the file downloads or uploads by corresponding Firefox instances. For private mode VEs there is no such binding and consequently these VEs can access only its own file system. Lastly the Perl script `safefox.pl` is associated with the SafeFox icons so that clicks on the icons trigger the execution of the script.

4.2. Secure URL redirection

As described in Section 3.2, a Firefox instance in the secure-book mark mode is created by clicking on an URL in the "Secure IC Bookmark" folder. Traffic with the designated secure site is redirected to the new Firefox instance running in a new VE. Participating components in this redirection process are shown in Figure 6. First, all SafeFox browser instances connect to the Internet through a Squid [23] Web proxy (in Figure 6, we show only the master Firefox). TCP connections not using the Web proxy port 3128 are blocked by the host firewalls, which is inaccessible and invisible for processes inside VEs. If the proxy setting of a browser instance is changed, either by the user or malware, the browser instance will not be able to reach the Internet.

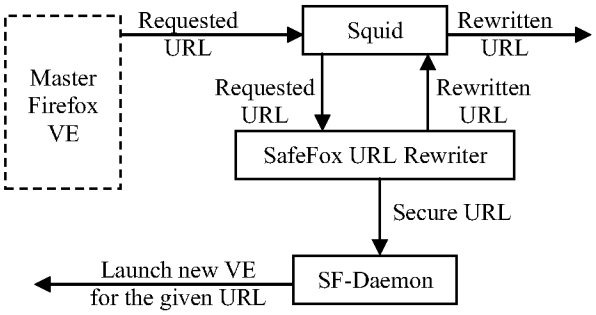


Figure 6: Secure bookmark redirection

Squid supports a feature called URL-rewrite. When this feature is enabled, it forwards every requested URL to a URL-rewriter program. We created an URL rewriter program, labeled "SafeFox URL Rewriter" in Figure 6, to perform the following functions. First, the rewriter checks the origin of the request. If the request does not come from the master Firefox (or precisely, its fixed IP address), then the rewriter simply returns the original URL to Squid, and the requesting Firefox instance obtains its desired contents. We do not support secure URL redirection in the private and secure-bookmark modes.

When the URL request originates from the master Firefox, the rewriter checks if the URL is a secure URL, that is, whether it is in the "Secure IC Bookmark" folder of the master Firefox (bookmarks in Firefox 3 can be found in the file `places.sqlite`). If requested URL is not in the folder, the rewriter returns the original URL to Squid and the master Firefox obtains its desired contents.

If the requested URL from the master Firefox *is* a secure URL, then the rewriter returns the URL http://127.0.0.1/redirect_messge.html to Squid. This request is served by a local Web server thttpd [34], standing for tiny htttpd due to its small memory footprint. The user will see in the master Firefox a message indicating a new secure-bookmark mode browser has been launched for the site (the message is not shown in the screenshots of Figure 5).

In the mean time, the SafeFox URL rewriter sends the requested secure URL to a backend daemon, labeled SF-Daemon in Figure 6, which in turn invokes `safefox.pl` to launch a new Firefox instance with the request secure URL as an input parameter. The new Firefox then makes a request of the URL through

Squid. Because the new Firefox VE is allocated an IP address different from the one of the master Firefox, its requests go through the SafeFox URL rewriter unchanged and it will subsequently obtain the contents of the designated secure URL.

4.3. SafeFox virtual appliance

After the SafeFox prototype was completed, we ported it to a VMware virtual appliance. The SafeFox VM is based on the server edition of Ubuntu 8.04 (for its small size) with its kernel replaced by our customized kernel. We use the same CentOS 5 template as in the Linux prototype. Shown in Figure 7 are the mixes of operating systems and virtualization layers when the SafeFox browsing environment is running on a Windows host as a virtual appliance. Lastly, we use Xming [35] as the X server on Windows hosts. The result is a mix of three operating systems and two virtualization layers as shown in Figure 7. Finding attack paths to penetrate such a heterogamous environment would be a formidable task.

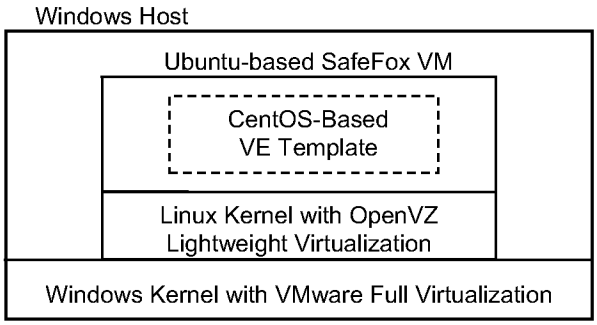


Figure 7: Heterogeneity with the SafeFox VM on Windows

5. Experiments and evaluation

In this section, we present experimental evaluation results of SafeFox. The evaluation contains two parts: 1) the overhead of disk and memory and startup delays and 2) the analysis of effectiveness of the SafeFox against common attacks for browsers.

5.1 Testbed and experiment design

We have described our Linux prototype of SafeFox in Section 4.1. Tests about SafeFox as a virtual appliance are conducted on a Dell Dimension 9200 workstation, with Intel Core2 processor (one core only) and 4GB memory. We will refer to the former as Testbed 1 and the latter Testbed 2.

We compare our lightweight approach of SafeFox with one which uses VMware VMs as replacements of VEs for host and session protection. The latter approach will be referred to as VVM. The VM used in the VVM is based on Ubuntu 8.04 server edition for its small size. We then install Firefox and related library. The result is a VM that takes 724 MB disk space. The VM is allocated with 128 MB memory, a very modest memory allocation for VMware virtual machines. For the fairness of comparison, we use a VMware feature called Link Clone to create multiple VMs. A link clone VM shares its virtual disk with the original VM. Changes in the file system of the cloned VM are kept in a difference file. This is similar to our way of constructing VEs which share one template.

5.2 Disk and memory overhead

First, we measure the overhead of hard disk. For the SafeFox, its CentOS template takes 586 MB. Approximately, a SafeFox VE consumes only 3 MB hard disk on the host, owing to its use of stack-able file systems to share a read-only OS template among all VEs. The small increases are caused by the COW operations of the Firefox instances running in the VEs.

For VVM, each VM will generate a .vmem file with the same size as the physical memory (128MB in this test) of the virtual machine and a new difference file with the starting size of 16MB. The size of the difference file will grow as the differences in the file systems between the cloned and original VM increase. Consequently it takes at least 144M (128+16) for each virtual machine. We then record the memory and disk usage of the two approaches with increasing numbers of VEs/VMs. The disk result of this test is shown in Figure 8. The X axis represents the numbers of browser instances running in VEs for SafeFox and in VMs for VVM. The Y axis shows the hard disk usage in MB. The diamond line represents for SafeFox results and square line for VVM. Clearly, SafeFox has much lower disk overhead than VVM. As seen in the figure, running less than 15 Firefox instances concurrently, most likely the common cases, causes negligible increases in disk usage with SafeFox.

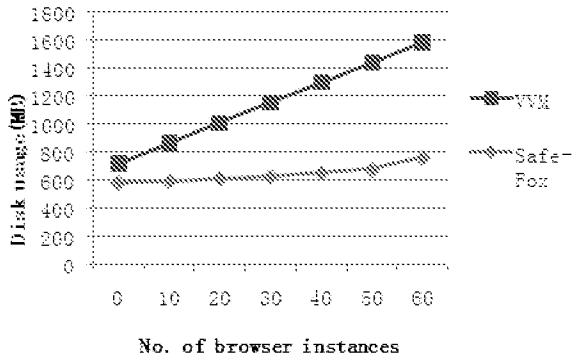


Figure 8. Disk space usage on the host

The memory usage test results are presented in Figure 9. For SafeFox, we stop the memory test when combined memory usage of Firefox/VE instances exceeds 2GB. As seen in the figure, it takes 58 instances to use half of the 4GB memory of Testbed 1. We point out that running 58 Firefox instances at a time is expected to be rare. In more realistic situations, SafeFox can support 15 concurrent Web sessions with approximate 500MB of memory overhead. For the VVM, we stop the test at 20 VMs, which consumed more than 2.5 GB of memory already. The results again prove the low overhead of the SafeFox browsing environment.

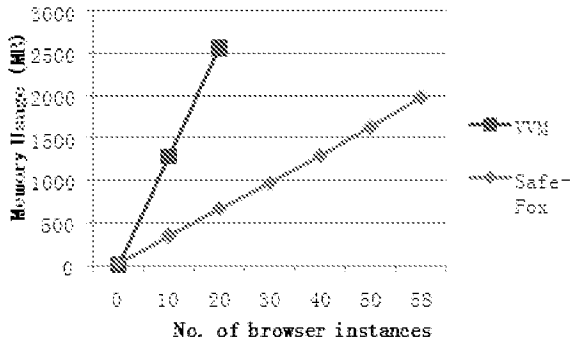


Figure 9. Memory usage on the host

5.3 Startup delay and redirection time for SafeFox appliance

Since less than 1% of client-end users use Linux, we expect most users will use our virtual appliance on Windows. Therefore, we measure the startup delay for virtual appliance on Windows. We run the SafeFox appliance and VVM on Testbed 2. The result is given in Table 1. We repeated every experiment three times and take the average. Cold-start means the browser instance is started for the first time after a reboot and the cache is not warm. Warm-start is the following starts. In addition, for the SafeFox appliance, VM is launched automatically each time the Windows is booted up. When the user wants to start a SafeFox browser, we only need to start a new VE and a Firefox inside the VM. The average time for SafeFox warm-start is 3.3s. The cold-start time is 12.3s. We also measured the startup delay for only starting a VE, which is 1.5s for warm start and 6.3s for clod start. Moreover, the redirection time -- time between the user hit the Web site in the secure bookmark folder and the secure browser shows up -- is 3.3s. This time is the same as the warm-start of a SafeFox instance, meaning that the overhead of Squid and SafeFox URL rewriter is negligible.

Table 1. Startup delay of browser instances

	Operation	Average Latency (s)
SafeFox	Warm-start	3.3
	Cold-start	12.3
	Redirection time	3.3
VVM	Warm-start	9.1
	Cold-start	23.2
native Firefox	Warm-start	1.5
	Cold-start	6.2

For VVM, the warm-start takes 9.1s, while the cold-start takes 23.2s. Both are slower than SafeFox. Note that we already optimized the starting of VVM by first taking a snapshot and then revert to it when starting VVM. The last two rows in the Table 1 shows the start delay for native Firefox browser instance running on Linux. The warm-start takes 1.5s and the cold-start takes 6.2s.

5.4 Network performance of SafeFox appliance

To measure network throughput of the SafeFox appliance, we use wget to download a 3GB large file and redirect the output to /dev/null on Linux or "nul" on Windows to remove the bottle neck of the relatively slow hard disk. The network card is a 1Gbit/s Ethernet card. We repeated the tests three times and averaged the result. Also, to minimize the effect of variability in network traffic between tests, the server and the client are both on a local network. The results are shown in Table 2. During the test, we only started one VE and one VM. VMtools is installed inside the testing VM for maximum performance. For SafeFox, we run wget inside the VE; for VVM, we run wget inside VM.

Table 2 Network performance

	throughput (download 3GB file) MB/s
SafeFox	88.0
VVM	86.2
Native host	90.0

From the Table 2, we can see that the throughputs of both SafeFox and VVM are close to that of the native host. The SafeFox appliance is slightly faster than VVM although it is also runs inside a VM. We attribute this to the fluctuation of the network.

5.5 Security analysis

In this section, we discuss the attacks that can be prevented by SafeFox, and the exceptions.

SafeFox is designed to provide *host* and *session* protection. The SafeFox appliance for Windows can be used to protect the Windows host and the sessions in the secure-bookmark mode. For *host* protection, SafeFox can defeat the typical drive-by downloads. A typical drive-by download exploits the vulnerabilities, such as [20,21,22] in the browser and downloads a file to the user's system and then executes it [17]. In SafeFox, all the browsers are running on a Linux VE. If the attacker succeeds in a drive-by download attack and can run some malware on the Linux VE, it will not affect the Linux host. Moreover, for the SafeFox appliance, the user's host is the Windows, if the attacker downloads malware targeting the Windows host, then he cannot run it in the Linux VE. In addition, Linux VE will be destroyed after the user exits the corresponding Firefox process. So the effective time for a Linux drive-by download exploit is depending on the user's behavior, which could be very short.

As for *session* protection, SafeFox is designed for protecting the secure bookmark browsers even if other browser instances (such as master or private browser) are compromised. Secure bookmark browsers reside in their own VEs and have their own metadata. Even if the attacker compromises a master or private browser instance, he still cannot get the sensitive information of the secure bookmark browsers. Unless OpenVZ layer is compromised, attacker can only access the files in the compromised master or private VEs. For example, the Gmail vulnerability in [2] will not work for SafeFox if the user puts the Gmail into the secure bookmark folder.

However, SafeFox is not designed for preventing "social engineering" attacks, which fool the user to download some malware and willingly execute it on the (Windows) host [17]. When using the SafeFox appliance, the user will have to download the executable to the shared folder on the Windows host, then run it to infect the Windows host. This set of manual actions shows intent on the user's part to run the software – and thus it is allowed as long as they have adequate privileges on the host. Also, if the secure Website itself is compromised or purposely designed with ill intention, it can compromise that particular instance of SafeFox, but not other running SafeFox instances.

The isolation protection of the SafeFox appliance depends on the two layers of virtualization – OpenVZ and VMware. If OpenVZ layer is compromised by the attacker, then the *session* protection is lost, meaning that other browser sessions can be compromised. But *host* protection still holds until and unless the VMware layer is also compromised. This raises the bar for the security because the attacker now need to compromise two different virtualization technologies on two different platforms: OpenVZ on Linux and VMware Player/Workstation on Windows. In addition, we also employ trust DNS service from OpenDNS [37]. However, if the DNS server is compromised, we cannot guarantee the user will land on the correct Web site. Finally, it is important to note that malware that successfully installs itself in the browser or breaks through the OpenVZ layer, will be extinguished when the browser is terminated and when the VM of the SafeFox appliance is terminated or reverted.

6. Related Work

This work leverages the results of several lines of prior research including virtualization and sandboxing applications. In this section we describe how we extend these works to provide strong isolation with high scalability.

Similar to our work, Cox et al [4] developed the Tahoma browser to protect Web applications. They use Xen virtualization to provide strong isolation between browser instances by running every instance in a separate Xen virtual machine. Our work differs in following ways. Firstly, with Tahoma, running N browser instances requires N Xen VMs. In contrast, SafeFox runs at most one (VMware) VM for the virtual appliance and uses OS-level virtualization to isolate multiple browser instances. Secondly, unlike Tahoma, we do not have to modify the browser source code. As a result, SafeFox can easily be extended to support any browser implementations on Linux, such as Opera or Konqueror. Our approach of the SafeFox appliance also has an important practical advantage: it does not require Windows users to switch to Linux. With a SafeFox virtual appliance, we bring to Windows users

the safety of Linux browsing [1], which is further enhanced by SafeFox.

NetTop [24] also use full virtualization to provide different VMs for multiple security levels. While it focuses on the general applications, we focus on browser applications and provide low virtualization overhead.

Without leveraging virtualization, processes provide stronger isolation than threads by running each of them in a dedicated virtual memory space. While many present popular browsers, such as Firefox and Internet Explorer 7, use threads to manage concurrent sessions, the recent trend is to create process-based architectures for strong browsing safety.

The OP browser [6], for instance, separates a browser into a number of subsystems, including a browser kernel, multiple Web page instances, network subsystem, user interface subsystem and storage subsystem. Each of them, even plug-ins, runs in its own process. This design provides a fine-grained isolation among different subsystems in a browser. In the same vein, Google Chrome [3] is comprised of two types of process components: a browser kernel process and a number of rendering engine processes (each for a Web site). Safety is further enhanced by limiting the privileges of the rendering engines. As a consequence, the vulnerabilities or executing ill-intended JavaScript's in rendering engines will not affect the browser kernel.

Internet Explorer 8 on Vista runs tabs in separate processes with different integrity levels [5]. It uses User Access Control mechanism on Vista and assigns different process integrities, such as high, medium and low to different processes. In addition, low and medium integrity tabs can reside in the same UI frame. This improves the reliability: rendering engine crashes do not result in full browser crashes.

Although process-architected browsers provide much stronger isolation and protection compared to the threads-based browsers, the inevitable IPC mechanisms used for process communication still provide the potential channels for taint propagation and system compromises.

There are also many projects developing techniques for securing web-based applications [25, 26, 27], securing JavaScript [28, 29, 30], protecting privacy [31, 32], and overcoming DNS rebinding attacks in browsers [33]. These projects are orthogonal to SafeFox.

[1] We acknowledge that browsing safety in Linux is not necessarily an inherit property of Linux. Possibly, it is due to the lack of interests in the hacker community. Either way, the consequence is the same.

7. Conclusion

We have described SafeFox, a safe and lightweight browsing environment. We have also ported it to a virtual appliance for Windows. We have shown how we use lightweight virtualization to provide *host* protection and *session* protection. In addition, SafeFox provides three browsing modes to support standard browser usage, secure transaction sites, and private browsing. Our experiments show that SafeFox has a lower overhead than a pure VM isolation solution that uses a VM for each Web site. Also, by separating secure browsers each into its own VEs, SafeFox has similar isolation properties as the approach based on full/para virtualizations but with much lower overhead.

8. References

[1] David Endler. The Evolution of Cross Site Scripting Attacks. Technical report, iDEFENSE Labs, 2002.

[2] New Tool to Automate Cookie Stealing from Gmail, Others, http://voices.washingtonpost.com/securityfix/2008/08/new_tool_automates_cookie_stea.html

[3] Barth, A., Jackson, C., and Reis, C. The Security Architecture of the Chromium Browser; http://crypto.stanford.edu/websec/chromium/chromium-security-architecture.pdf.

[4] Richard Cox, Steven Gribble, Henry Levy, and Jacob Hansen. A safety-oriented platform for Web applications. In Proceedings of the 2006 IEEE Symposium on Security and Privacy, Washington, DC, May 2006.

[5] Andy Zeigler. IE8 and Loosely-Coupled IE, March 2008. http://blogs.msdn.com/ie/archive/2008/03/11/ie8-and-loosely-coupled-ie-lcie.aspx.

[6] Chris Grier, Shuo Tang, and Samuel T. King. Secure web browsing with the op web browser. In IEEE Symposium on Security and Privacy, 2008.

[7] VMware, http://www.vmware.com.

[8] Paul Barham , Boris Dragovic , Keir Fraser , Steven Hand , Tim Harris , Alex Ho , Rolf Neugebauer , Ian Pratt , Andrew Warfield, "Xen and the Art of Virtualization," Proceedings of the nineteenth ACM symposium on Operating systems principles, October 19-22, 2003, Bolton Landing, NY, USA.

[9] KVM (Kernel-based Virtual Machine). http://kvm.qumranet.com/kvmwiki/Front_Page.

[10] Jon Watson. VirtualBox: bits and bytes masquerading as machines. Linux Journal, Volume 2008, Issue 166, February 2008.

[11] Poul-Henning Kamp and Robert N. M. Watson. Jails: Confining the omnipotent root. In Proc. of 2nd Intl. SANE Conference, May 2000.

[12] Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., and Peterson, L. "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," In Proceedings of the ACM Sigops/Eurosys European Conference on Computer Systems (EuroSys '07), Lisbon, Portugal, March 2007.

[13] Osman, S., Subhraveti, D., Su, G., and Nieh, J. 2002. The design and implementation of Zap: a system for migrating computing environments. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation. (Boston, Massachusetts, December 09 - 11, 2002). OSDI '02.

[14] Price, D. and Tucker, A. 2004. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In Proceedings of the 18th USENIX Conference on System Administration (Atlanta, GA, November 14 - 19, 2004).

[15] OpenVZ server virtualization open-source project. http://openvz.org.

[16] Wright, C. P., Dave, J., Gupta, P., Krishnan, H., Quigley, D. P., Zadok, E., and Zubair, M. N. 2006. Versatility and Unix semantics in namespace unification. Trans. Storage 2, 1 (Feb. 2006), 74-105.

[17] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The ghost in the browser analysis of web-based malware. In Proceedings of the 2007 Workshop on Hot Topics in Understanding Botnets (HotBots), April 2007.

[18] See the man page of the `mount` command.

[19] Yih Huang, Angelos Stavrou, Anup K. Ghosh and Sushil Jajodia., Efficiently Tracking Application Interactions using Lightweight Virtualization, In the Proceeding of the 1st Workshop on Virtualization Security (VMSec), in conjunction with ACM CCS 2008, October 2008.

[20] Mitre. CVE-2006-7228, 2006.

[21] Mitre. CVE-2007-3743, 2007.

[22] Mitre. CVE-2008-3360, 2008.

[23] Squid, http://www.squid-cache.org/

[24]NetTop,http://www.vmware.com/pdf/TechTrendNotes.pdf

[25] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng. Secure web applications via automatic partitioning. In Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP), Oct 2007.

[26] S. Chong, K. Vikram, and A. C. Myers. Sif: Enforcing confidentiality and integrity in web applications. In Proceedings of the 16th Usenix Security Symposium, Boston, MA, USA, August 2007.

[27] U. Erlingsson, B. Livshits, and Y. Xie. End-to-end web application security. In Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS XI), May 2007.

[28] T. Jim, N. Swamy, N. Swamy, and M. Hicks. Defeating script injection attacks with browser-enforced embedded policies. In Proceedings of the16th international conference on World Wide Web, 2007.

[29] C. Reis, J. Dunagan, H. Wang, O. Dubrovsky, and S. Esmeir. Browsershield: Vulnerability-driven filtering of dynamic html. In Proceedings of The 7th Symposium on Operating Systems Design and Implementation (OSDI), November 2006.

[30] D. Yu, A. Chander, N. Islam, and I. Serikov. Javascript instrumentation for browser security. In Proceedings of the 34th annual ACM SIGPLANSIGACT symposium on Principles of programming languages, New York, NY, USA, January 2007.

[31] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In Proceedings of the 15th international conference on World Wide Web (WWW). ACM Press, 2006.

[32]U. Shankar and C. Karlof. Doppelganger: Better browser privacy without the bother. In Proceedings of the 13th ACM conference on computer and communications security (CCS), 2006.

[33] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh. Protecting browsers from dns rebinding attacks. In Proceedings of the 14th ACM conference on computer and communications security (CCS), 2007.

[34] thttpd. http://freshmeat.net/projects/thttpd/

[35] Xming http://www.straightrunning.com/XmingNotes/

[36] Browser Appliance http://www.vmware.com/appliances/directory/815 Aug, 2008

[37] OpenDNS. http://www.opendns.com